ClickHouse at Twilio SendGrid

August 2024

Sam Nguyen, Principal Engineer

Agenda

- What is Twilio SendGrid (TSG)?
- Why ClickHouse?
- The Proof of Concept
- Future of ClickHouse @ TSG

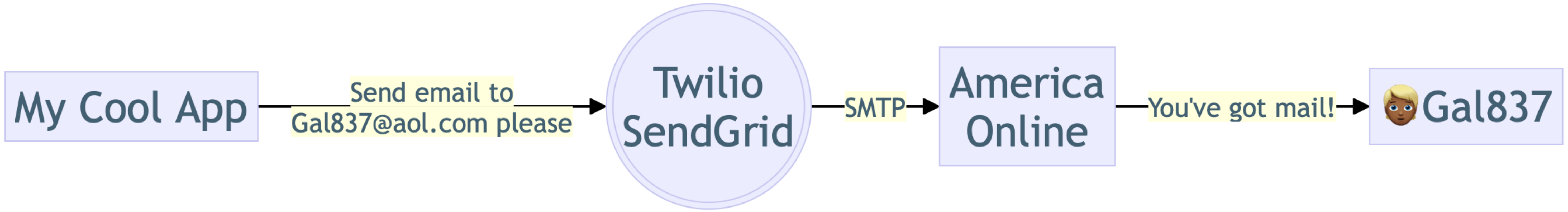
Goal: provide data points for others who are early in their ClickHouse journey

About me

- At Twilio SendGrid since 2013
- New to ClickHouse and data engineering

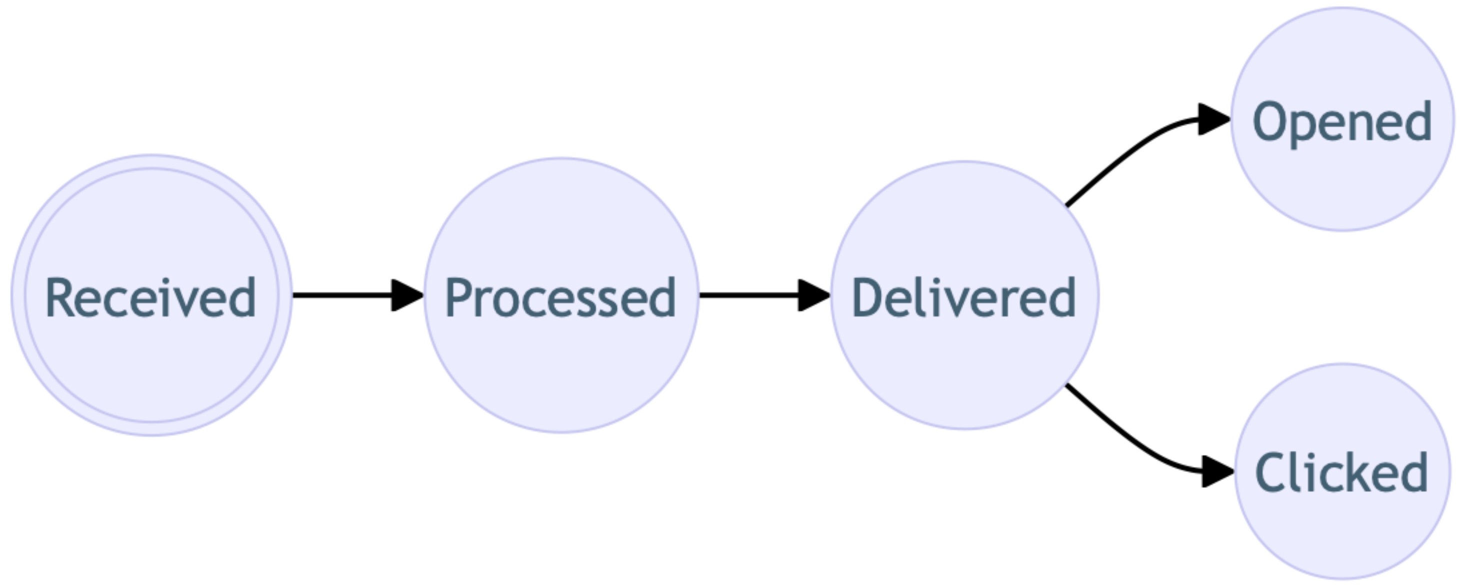
What is Twilio SendGrid (TSG)?

- A way for your app to send email



- Scale: 10 billion emails/day
- Peaks: 1-2 million events/sec

What events does TSG generate?

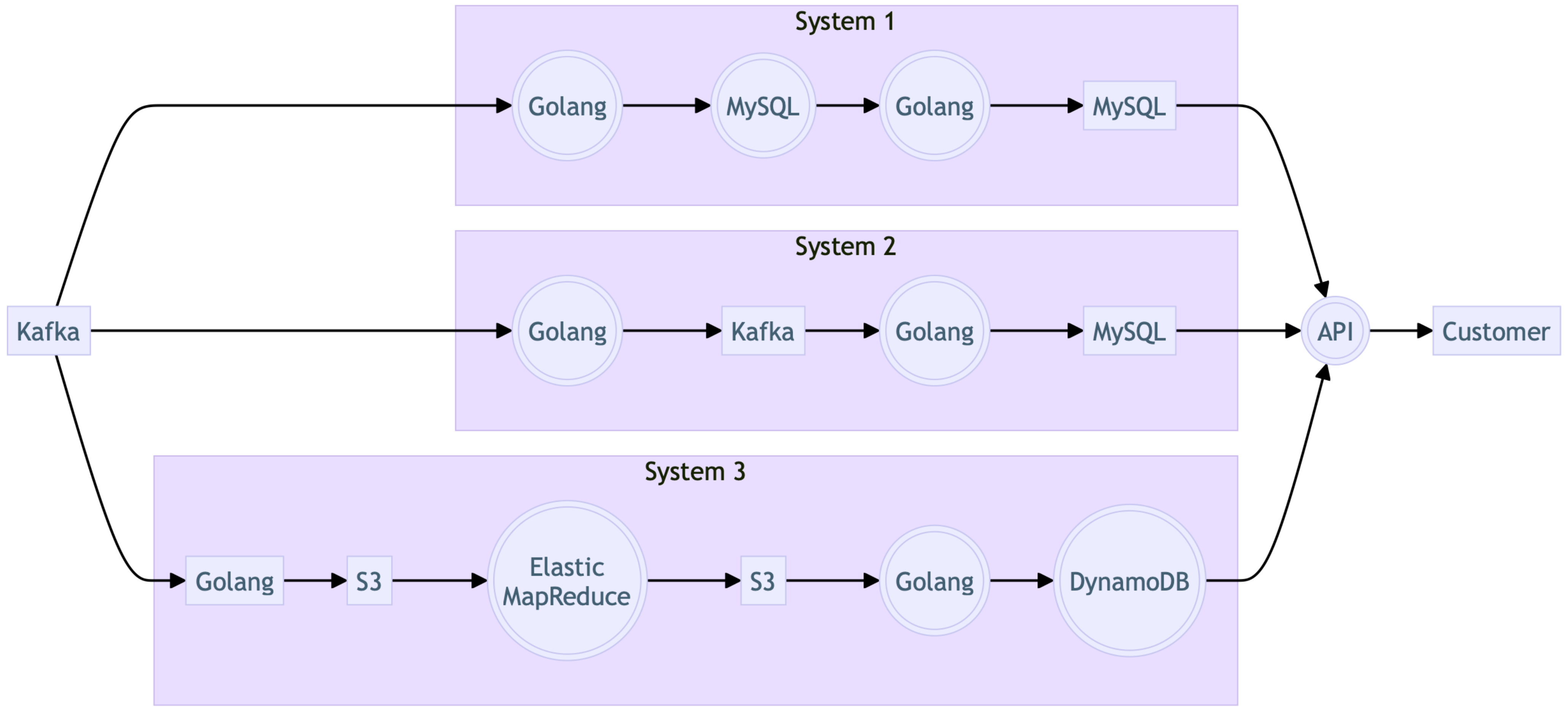


Business problems to solve

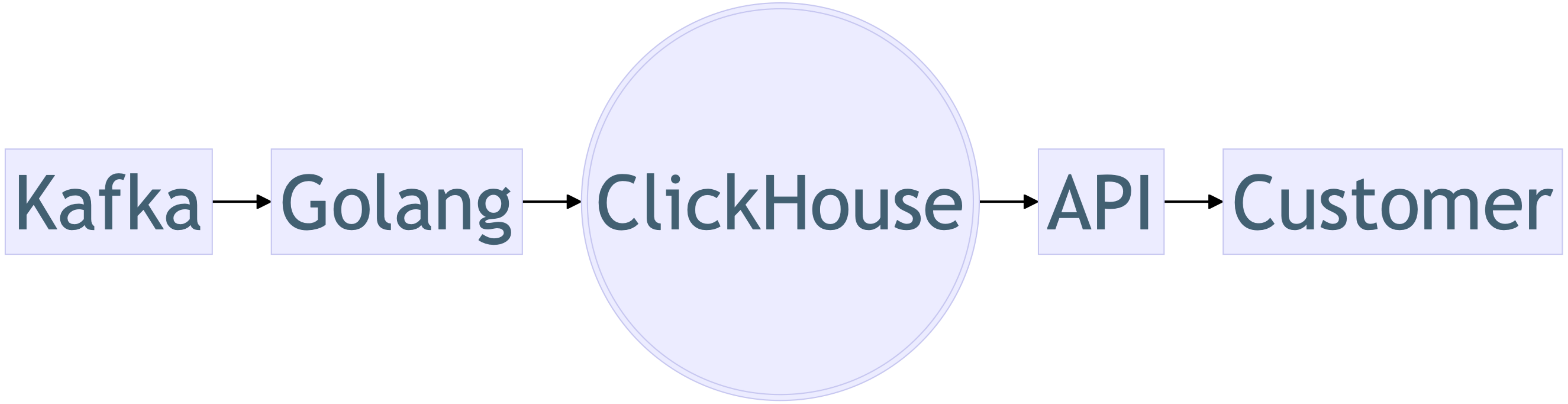
- Customer-facing Analytics (email campaign performance)
  - Fixed slicing dimensions
- Billing Analytics
  - High accuracy, subject to Sarbanes-Oxley
- Customer-facing Logs (detailed troubleshooting)
  - 30 day retention (~150TB)
  - Fixed filtering fields
- 1-2 million events per second

Incidental problems to solve

- Consolidation of tech stack 🍝



Target architecture



Goal: 15 year service life

Why ClickHouse

- Previous team already did technology evaluation and chose ClickHouse for analytics use case
- Current leadership had positive sentiment on ClickHouse
- ClickHouse Cloud enabled a fast POC phase
- Consolidation of tech stack surface area

The Proof of Concept

1. Measure ingestion performance
2. Measure query performance
3. Deliver recommendation to leadership

Proof of Concept:
Ingesting data into ClickHouse

1. S3 table engine (one-time ingestion)
2. Custom Kafka Consumer (continuous ingestion)

Measuring ingestion performance

```sql
-- Materialized view that keeps
-- the latest event timestamp from each region
CREATE TABLE latest_timestamps
(
  region String,
  latest_timestamp AggregateFunction(max, DateTime)
) ENGINE = AggregatingMergeTree

CREATE MATERIALIZED VIEW latest_timestamps_mv TO latest_timestamps
AS SELECT region, maxState(timestamp) as latest_timestamp
FROM raw_events
GROUP BY region
```

Current performance: < 1 minute freshness

Measuring query performance

Replaying API requests

```
{
    "event": "http_request",
    "method": "GET",
    "path": "/v1/messages",
    "query": "email=Ga1837@aol.com",
    "http_status": 200,
    "latency_sec": 1.234
}
```

Replaying API requests

Emitting structured log output

```
{
    "event": "http_request_replay",
    "method": "GET",
    "path": "/v1/messages",
    "query": "email=Gal837@aol.com",
    "original_latency_sec": 1.234,
    "clickhouse_latency_sec": 0.1234
}
```

Analyzing query performance (existing)

```
$ clickhouse local

WITH round(latency_sec) AS seconds
SELECT seconds, count(*), bar(count(*), 0, 100000, 24)
FROM file('replay-results.json')
GROUP BY seconds

    ┌─seconds─┬─count()─┬─bar(count(), 0, 100000, 24)─┐
 1. │       1 │   61070 │ ██████████████▋             │
 2. │       2 │   30926 │ ███████▍                    │
 3. │       3 │   10391 │ ██▍                         │
 4. │       4 │    5701 │ █▍                          │
 5. │       5 │    4587 │ █                           │
 6. │       6 │    3310 │ ▊                           │
 7. │       7 │    2536 │ ▌                           │
 8. │       8 │    1820 │ ▍                           │
 ...
50. │      50 │       2 │                             │
    └─seconds─┴─count()─┴─bar(count(), 0, 100000, 24)─┘
```

Analyzing query performance (ClickHouse)

```
$ clickhouse local

WITH round(latency_sec) AS seconds
SELECT seconds, count(*), bar(count(*), 0, 100000, 24)
FROM file('replay-results.json')
GROUP BY seconds

    ┌─seconds─┬─count()─┬─bar(count(), 0, 100000, 24)─┐
 1. │       0 │   48536 │ ███████████▋                │
 2. │       1 │   77210 │ ██████████████████▌         │
 3. │       2 │     635 │ ▏                           │
 4. │       3 │     118 │                             │
 5. │       4 │      42 │                             │
 6. │       5 │      13 │                             │
 7. │       6 │      10 │                             │
 8. │       7 │       7 │                             │
 9. │       8 │       2 │                             │
10. │       9 │       2 │                             │
    └─────────┴─────────┴─────────────────────────────┘
```

Extra stuff to figure out

- User & credential management (no IAM-like solution)
- Schema migrations (golang-migrate)
- Exporting arbitrary Prometheus metrics (burningalchemist/sql_exporter)

Future of ClickHouse at Twilio SendGrid

1. Finish the POC!
2. Implement new customer-facing features
3. Migrate all existing analytics and logs functionality to ClickHouse
4. Provide other teams with patterns for using ClickHouse in their systems

Learning Resources

- Podcast Interviews
- ClickHouse Release Webinars
- ClickHouse Training
- Load data into `clickhouse local` and query it
- Carnegie Mellon Advanced Course on OLAP Databases
- ClickHouse Account Team

Questions?